Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Mihelj

Primerjava razvoja prave domorodne mobilne aplikacije in razvoja z uporabo ogrodja React Native

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Alenka Kavčič

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Primerjava razvoja prave domorodne mobilne aplikacije in razvoja z uporabo ogrodja React Native

Tematika naloge:

V okviru diplomske naloge primerjajte dva pristopa k razvoju mobilne aplikacije: izdelava prave domorodne javanske aplikacije in izdelava aplikacije s pomočjo ogrodja React Native. Primerjajte tako sam postopek razvoja mobilne aplikacije kot tudi rezultat razvoja, to je izdelano aplikacijo. V ta namen zasnujte in implementirajte preprosto mobilno aplikacijo, ki uporablja vsaj enega od razpoložljivih senzorjev mobilne naprave. Pripravite ustrezne kriterije, na podlagi katerih primerjate oba pristopa in razviti aplikaciji. Opišite tudi glavne prednosti in slabosti vsakega od omenjenih pristopov.

Najlepša hvala staršem, mentorici in Nini za potrpljenje med pisanjem diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
JS	Javascript	Javascript
RN	React Native	React Native
NPM	Node Package Manager	paketni upravitelj Node.js
SDK	Software Development Kit	paket za razvoj programske opreme
APK	Android Package	paket aplikacije za Android
ADT	Android Developer Tools	razvojna orodja za Android
XML	Extensible Markup Language	razširljiv označevalni jezik
HTML	Hypertext Markup Language	jezik za označevanje nadbesedila
CSS	Cascading Style Sheet	prekrivni slogi
JSX	Syntax Extension to Javascript	sintaktična razširitev Javascripta
DOM	Document Object Model	objektni model dokumenta
GPS	Global Positioning System	sistem za globalno pozicioniranje
API	Application Programming Interface	programski vmesnik
REST	Representational State Transfer	arhitekturni stil REST
JSON	Javascript Object Notation	opis objekta za Javascript
URL	Uniform Resource Locator	enotni naslov vira
MB	Megabyte	megabajt

Povzetek

Naslov: Primerjava razvoja prave domorodne mobilne aplikacije in razvoja z uporabo ogrodja React Native

Avtor: David Mihelj

V diplomski nalogi sta primerjana orodje za domorodni razvoj in ogrodje React Native za razvoj mobilne aplikacije z vidika razvijalca in uporabnika. Na razvojni strani so upoštevana količina orodij, njihove sposobnosti in preprostost uporabe, kar vpliva na čas razvoja, kvaliteto kode in funkcionalnost aplikacije. Z uporabnikovega stališča pa so primerjane funkcionalnosti aplikacije, uporabniška izkušnja in odzivnost. Razlike za uporabnike so v večini primerov zanemarljive ali neopazne, saj React Native dosega sposobnosti domorodnega razvoja. Večje razlike občutijo razvijalci, veliko pa je odvisno od tipa projekta in razvojne ekipe, najbolj zaradi različnega programskega jezika (Java za domorodne aplikacije, Javascript za React Native), zaradi katerega so potrebna drugačna razvojna okolja in znanje drugega jezika.

Ključne besede: Android, aplikacija, mobilno, domorodno, React Native.

Abstract

Title: Comparison of real native and React Native mobile application development

Author: David Mihelj

This diploma compares React Native with native tools for developing mobile applications from the perspective of both the developer and user. Factors that impact project duration, quality of code and functionality of the final product like number of tools, their capabilities and simplicity of use have been considered for the development aspect. Functionality, user experience and responsiveness were assessed from the user's perspective, however they are mostly unnoticeable or non-existent due to the maturity of React Native. The biggest difference is felt by developers because of different programming languages used (Java for native and Javascript for React Native applications), however it does depend on the type of project and development team.

Keywords: Android, native, mobile, application, React Native.

Poglavje 1

Uvod

Mobilni telefoni so v zadnjih desetih letih postali nepogrešljiv spremljevalec. Klicanje in sporočanje sta postali samo dve izmed funkcij, ki jih ponuja naprava, za ostalo se zanašamo na aplikacije. Za razvoj ima programer več možnosti. Poleg domorodnega razvoja obstaja še veliko orodij, ki jih je ustvarila skupnost, vsako pa ima določene lastnosti, ki jih naredijo privlačne za različne tipe razvijalcev in projektov. Trenutno popularno ogrodje za razvoj mobilnih aplikacij React Native (RN) bomo primerjali z domorodnim razvojem s pomočjo aplikacije Voznikov pomočnik (*ang. Driver Assistant*). React Native je izšel iz projekta React, ki je ogrodje za spletne vmesnike, zato tudi uporablja spletne tehnologije (Javascript (JS) in jezike, podobne HTML in CSS). Domorodne aplikacije pa so napisane v Javi. Domorodne aplikacije imajo nizkonivojski dostop in omogočajo boljšo optimizacijo, kar je pomembno za zahtevnejše aplikacije. Hkrati je razvoj tudi bolj zapleten. React Native poskuša biti preprostejši, hkrati pa ponuja tudi razvoj za več platform hkrati. Načeloma to pomeni, da so domorodne aplikacije lahko hitrejše zaradi optimizacije, ni pa nujno.

Primerjava je razdeljena na dva dela. Razvojni del je razdelan po orodjih, ki se uporabljajo pri posameznem načinu razvoja in komponentah aplikacije. Predstavljena so orodja, ki so bila uporabljena za izdelavo posamezne aplikacije, hkrati pa so še naštete razlike, ki razvijalcu pomagajo pri izbiri. Pri

predstavitvi razvoja je za posamezno komponento najprej razložen postopek implementacije, nato pa so, glede na opisane razlike, podani razlogi za izbiro določenega načina razvoja z upoštevanjem potreb projekta in zahtev razvijalcev. Primerjani sta tudi končni aplikaciji, po kriterijih, ki med njima čim bolj izpostavijo razlike.

Poglavje 2

Tehnologije in orodja

2.1 Razlika med ReactJS in React Native

ReactJS je knjižnica za Javascript, namenjena izdelavi vmesnikov spletnih aplikacij, ki izkorišča princip model - pogled - upravljalec (ang. *Model – View – Controller*). Izdelali so jo pri Facebooku, njena glavna prednost pa je konstrukt z imenom stanje (ang. *state*), ki omogoča osveževanje podatkov brez ponovnega nalaganja spletne strani. Stanje je poseben konstrukt, podoben globalni spremenljivki, uporablja pa se za podatke, ki se pogosto spreminjajo in se jih prikazuje na zaslonu. V drugih programskih jezikih mora razvijalec za prikaz sprememb poskrbeti sam, na primer z uporabo povratnih klicev. ReactJS po spremembi podatkov v stanju sam poskrbi za osvežitev vmesnika. Iz te osnove je izpeljan React Native, ki omogoča izdelavo mobilne aplikacije z uporabo arhitekture ReactJS. React Native v ozadju prevaja Javascript v domorodno kodo Java, ki se izvaja na napravi.

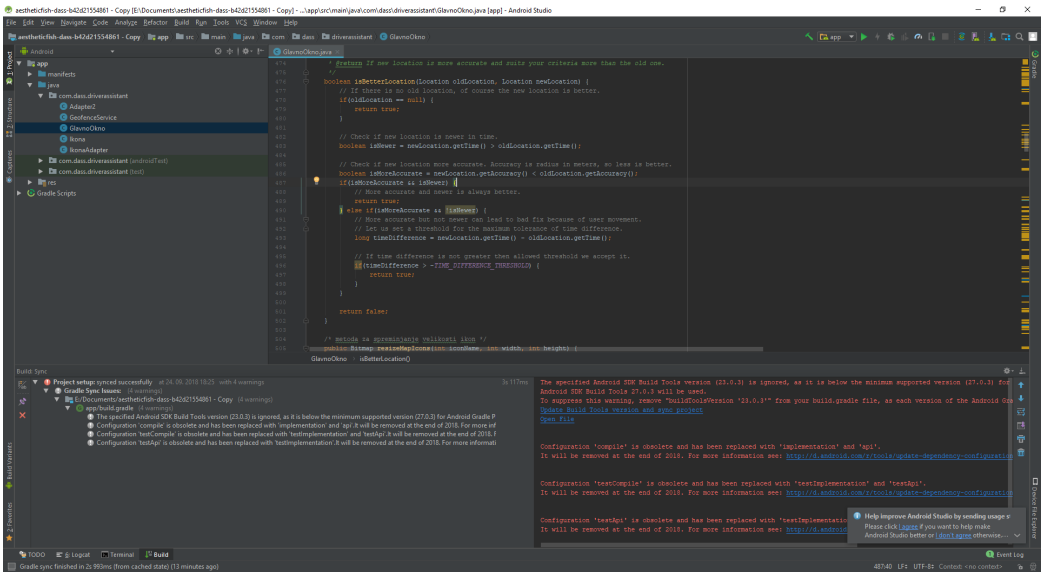
2.2 Programski jezik

Primerjana načina razvoja uporabljata različna programska jezika. Domorodne aplikacije se razvija v Javi, React Native pa je ogrodje za razvoj aplikacije z Javascriptom. Jezika sta si v nekaterih pogledih podobna in v nekaterih različna. V obeh jezikih se vidi vpliv C in C++ (deklaracija spremenljivk, objekti, klicanje metod ...), v podrobnostih pa se razlikujeta. JS je preprostejši za neizkušene programerje, saj ima bolj fleksibilno sintakso, na primer ločevalni znaki na koncu stavkov niso nujni, niti niso nujni zamiki vrstic kot pri Pythonu. Spremenljivke nimajo določenih tipov, nadzor dostopa ima drugačna pravila, podpira razrede in objekte, vendar ima slabšo podporo itd. Java je v teh pogledih ravno nasprotna, posamezni ukazi morajo biti ločeni s podpičjem, spremenljivke imajo določen tip, Java pa ima tudi odlično podporo za objekte in razrede. Zaradi teh olajšav je JS preprostejši za uporabo, vsaj za neizkušene programerje, če pa razvijalec že pozna katerega izmed jezikov, bo verjetno lažje delal z znanim jezikom. Glede hitrosti je težko zagotovo trditi, kateri jezik je hitrejši zaradi različnih načinov uporabe. Javascript se večinoma uporablja pri spletnih tehnologijah, na strežnikih in brskalnikih, Java pa za samostojne aplikacije na različnih napravah. Hitrost je tudi zelo odvisna od optimizacije programa, kar je povezano z izkušenostjo programerja. Razlika se lahko pokaže še v sprotnem prevajalniku, za Javo je starejši in mogoče bolj optimiziran kot JS tolmači, ali pa v hitrosti izvajanja RN aplikacije, ki predstavlja dodaten sloj med Javo in JS.

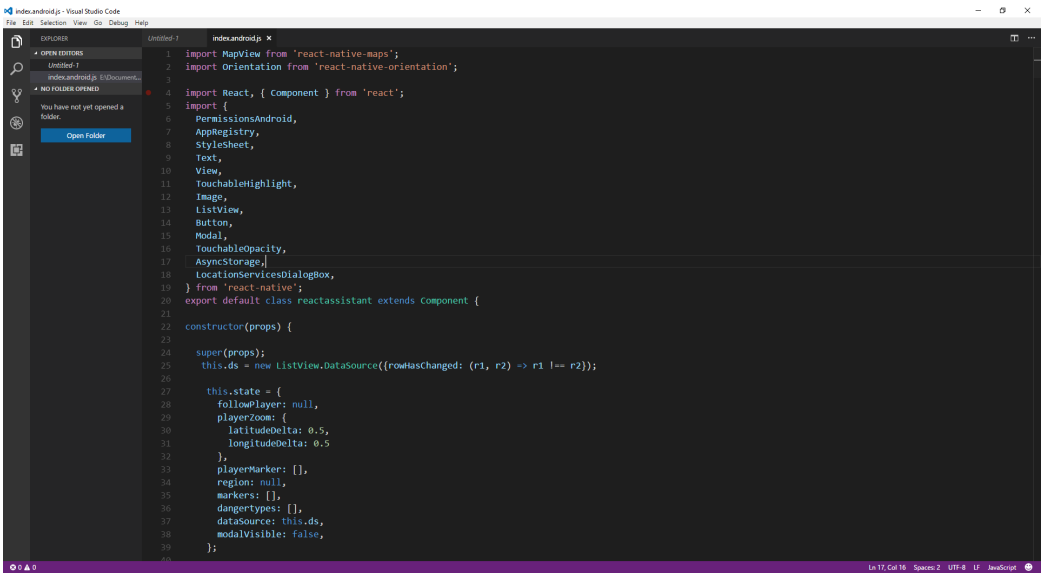
2.3 Urejevalnik kode

Ker primerjana načina razvoja uporabljata različna programska jezika, so različni tudi urejevalniki kode. Za domorodni razvoj je na voljo namenski program Android Studio. Osnova je urejevalnik kode IntelliJ, ki ima dodana orodja za razvoj aplikacij Android, na primer posnemovalnik (ang. *emulator*) naprav Android, pomočnik za nalaganje in posodabljanje razvojnih orodij itd. Vsebuje tudi bližnjice za hiter dostop do pogosto uporabljenih funkcij.

React Native nima takšne podpore, saj je samo skupek orodij in knjižnica za Javascript. Izdelovalci niso izdelali namenskega grafičnega orodja, zato mora razvijalec najti alternativo, ki mu najbolj ustreza. Največkrat omenjeni programi so Atom, Visual Studio Code, Sublime in Webstorm. Prva dva sta izdelana na isti osnovi, njuna filozofija pa je preprostost, kompaktnost in razširljivost, podobna sta si tudi vizualno. Sta novejša od ostalih dveh. Sublime je še bolj preprost, saj je v osnovi samo urejevalnik teksta, Webstorm pa je med vsemi najbolj zapleten, vendar tudi močan, saj spada med integrirana razvojna okolja (ang. *Integrated Development Kit* ali IDE). Najbolj opazna razlika med Android Studiem in ostalimi programskimi okolji so manjkajoča orodja za razvoj aplikacije Android, na primer vdelan Software Development Kit (SDK) in posnemovalnik. Razlika med Android Studiem in Visual Studio Code je vidna na slikah 2.1 in 2.2. Poleg hierarhije datotek in programske kode, ki jih prikazujeta oba, so v Android Studiu vidna tudi statusna okna, povezana z razvojem aplikacij, ponuja pa tudi bližnjice do dodatnih orodij.



Slika 2.1: Privzeti pogled v Android Studiu.



Slika 2.2: Privzeti pogled v Visual Studio Code.

Poglavje 3

Aplikacija

Kot redni udeleženci v prometu smo razmišljali, da je telefon premalo izkoriščen med vožnjo. Velika prednost mobilnih telefonov je njihova vseprisotnost in povezanost v splet. Aplikacije za pomoč v prometu že obstajajo, na primer Google Maps, ki opozarjajo na zastoje in zaprte ceste, AMZS prikazuje dogodke, ki jih spremlja DARS itd. Nismo pa do sedaj zasledili nobene, ki bi prikazovala manjše dogodke in nevarnosti, ki niso dovolj moteči, da bi ovirali promet, vendar so vseeno nevarni za voznike, kot na primer kamenje na cesti, ustavljen avto v nepreglednem ovinku, zamrznjeno cestišče in tako naprej. Na takšne nevarnosti lahko najhitreje opozorijo vozniki, ki so se jim izognili in s tem pripomorejo k večji varnosti drugih s pomočjo naprave, ki je skoraj vedno z nami.

Ker se aplikacijo uporablja tudi med vožnjo, mora biti zelo preprosta in nemoteča, da ne ogroža varnosti na cesti. Glavni element je karta, na kateri se prikazuje lokacija uporabnika (če dovoli uporabo lokacije). Na karti so prikazane nevarnosti v obliki prometnih znakov, ki predstavljajo tip nevarnosti (kamenje, spolzko ali poledenelo cestišče, prometna nesreča, zastoj, generična nevarnost, policija ...). Na zaslonu je še nekaj gumbov, med drugim za vnašanje nevarnosti, po kliku se shrani trenutna lokacija, uporabnik pa izbere tip nevarnosti. Po izbiri je nevarnost takoj vidna na glavnem zaslonu, njeni podatki pa se preko spleta prenesejo v strežnik, od koder jo pridobijo

še ostali uporabniki. Vsaka nevarnost ima tudi oceno, zato da se nepravilne odstrani, resnične nevarnosti pa ostanejo. Gumbi za oceno se prikažejo, ko je razdalja dovolj majhna. Preostala gumba sta še za nastavitve, kjer se lahko nastavi tip karte (satelitska slika, ceste ali hibrid), in pa stikalo za sledenje lokaciji. V aplikaciji je čim manj gumbov, da si je lažje zapomniti njihove položaje, poleg tega pa so večji, da jih je lažje zadeti. Postavljeni so narazen in stran od sistemskih gumbov, da bi se uporabnik čim manjkrat zmotil. Dokončana aplikacija je vidna na sliki 3.2.

Aplikacija mora torej omogočati:

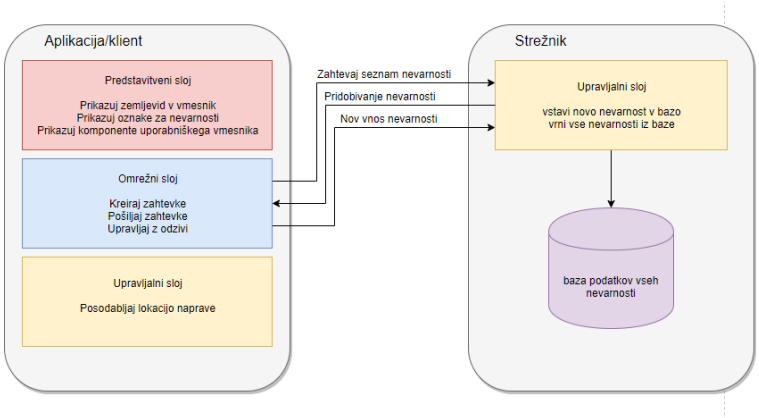
- prikaz karte
- prikaz uporabnikove lokacije na karti
- prikaz nevarnosti na karti
- vnašanje novih nevarnosti
- ocenjevanje nevarnosti ...

Te funkcionalnosti za normalno delovanje potrebujejo tudi zunanje senzorje in vhodno-izhodne naprave. Mobilno omrežje ali brezžični internet se uporablja za nalaganje karte, komunikacijo z glavnim strežnikom za izmenjavo podatkov o nevarnostih (pridobivanje s strežnika, nalaganje novih in ocenjevanje), pa tudi za določanje lokacije. Za bolj natančno določanje pozicije se uporablja antena GPS. Drugih senzorjev aplikacija ne uporablja.

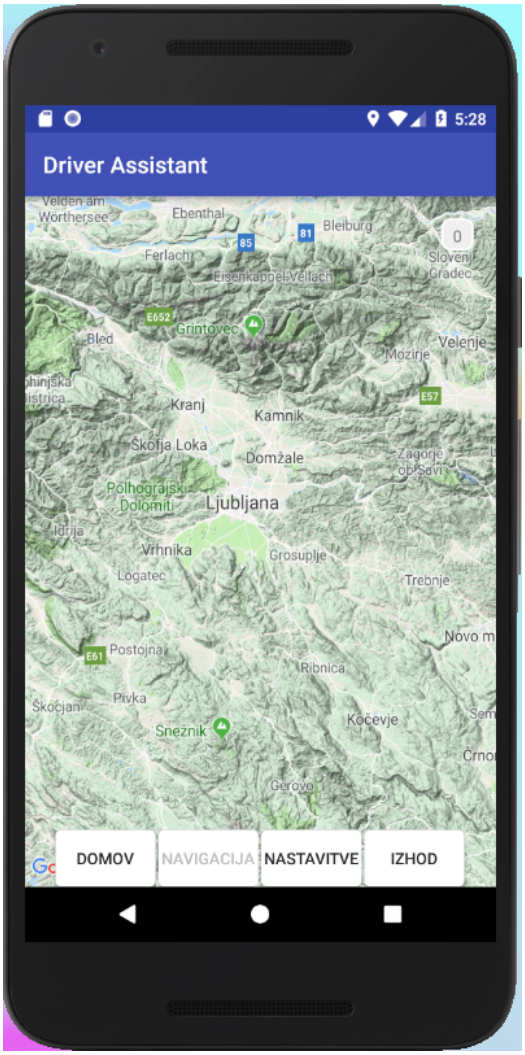
Naprave si nevarnosti izmenjujejo preko glavnega strežnika. Naprave strežniku pošiljajo nove nevarnosti in zahteve za pridobitev nevarnosti, ki se pošlje po pretečenemu času oziroma razdalji. V zahtevi se pošlje tudi trenutna lokacija, zato da strežnik ne pošilja čisto vseh nevarnosti, ampak samo najbližje. Komunikacija poteka preko protokola Representational State Transfer (REST). REST zahtevek ima obliko naslova URL, kjer je določen spletni naslov in dodatni argumenti. Strežnik na podlagi teh podatkov vrne podatke v zahtevani obliki, največkrat XML ali JSON.

Ker aplikacija ni zelo kompleksna, saj le pošilja in prejema podatke s strežnika in jih prikazuje, je tudi struktura obeh programov precej preprosta. Programa sta zapisana v le eni datoteki in enem razredu (JS samo delno podpira razrede), posamezne funkcionalnosti pa so implementirane v funkcijah. Ker je React Native namenjen ustvarjanju spletnih vmesnikov, se koda vmesnika precej bolj prepleta s programom kot v domorodnem programu, kjer je potrebnih več korakov za spreminjanje vmesnika, do katerega dostopa le posredno.

V grobem je struktura aplikacije sestavljena s treh delov, predstavitvenega, omrežnega in upravljalnega. Skica je na sliki 3.1. Upravljani del skrbi za delovanje aplikacije in usmerja tudi druge komponente. Omrežni del komunicira s strežnikom in opravlja tri naloge - na strežnik pošilja zahtevke (za nove nevarnosti in za pridobivanje nevarnosti), sprejema odzive strežnika in po sprejemu sproži postopke za obdelavo podatkov. Uporabniku najbližji je predstavitveni sloj, ki prikazuje uporabniški vmesnik, osvežuje pogled na zemljevidu, prikazuje prejete informacije in oddaja ukaze upravljalnemu delu. Strežnik ima samo dva sloja, podatkovno bazo, ki drži vse nevarnosti ter upravljalni sloj, ki sprejme zahtevke z mobilnih naprav in v podatkovno bazo shrani nove nevarnosti ali pa napravi pošlje nazaj seznam nevarnosti, ki izpolnjujo pogoje.



Slika 3.1: Skica arhitekture aplikacije.



Slika 3.2: Domorodna aplikacija.

Poglavje 4

Primerjava

Primerjava je razdeljena na dva glavna dela: razvojni in uporabniški. Razvojni del je sestavljen iz primerjave orodij in primerjave komponent aplikacije. Takšna razdelitev je primerna, ker je blizu razvojnemu procesu, zato jo razvijalec hitro razume in hkrati neposredno primerja uporabo in rešitev obeh načinov razvoja. Primerjava je izvedena na način, da se ločeno predstavi način uporabe, na koncu pa direktno primerja, izpostavi razlike in predlaga pravo izbiro z upoštevanjem različnih okoliščin razvoja. Ta postopek najprej predstavi delovanje, zato da ima bralec vsaj osnovno predstavo o uporabi, hkrati pa si lahko že sam ustvari mnenje. Na koncu pa je še direktna primerjava, kjer so izpostavljene razlike, ki jih bralec mogoče ni opazil.

Uporabniška primerjava je krajša, ker so razlike manjše. V tem poglavju so primerjane dokončane aplikacije, njihova velikost, odzivnost, razlike v funkcionalnosti in uporabi. Kriteriji so bile različne meritve (odvisno od tipa podatkov, velikost ali čas itd.), v komentarjih pa so podani razlogi za in proti izbiri RN.

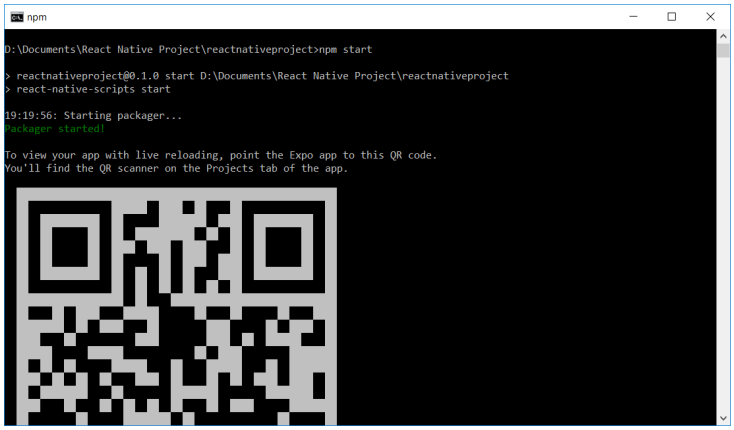
4.1 Namestitev orodij

Za razvoj so potrebne tri zadeve - urejevalnik kode, Android SDK in posnemovalnik za testiranje. Za razvoj domorodne aplikacije se lahko uporabi Android Studio, ki vsebuje vse potrebno, ali pa kakšen drugi urejevalnik kode Java (npr. Eclipse), je pa potem treba posebej naložiti SDK in posnemovalnik ter konfigurirati projekt, da omogoča funkcije, ki jih ponuja Studio že privzeto. Tako Android Studio kot Eclipse (najbolj uporabljana IDE-ja za domorodni razvoj) ponujata grafični naložitveni paket, ki ima podoben potek ostalim programom.

Ravno tako je Android SDK potreben pri razvoju z React Native. Najlažje se ga pridobi hkrati z Android Studiem, ki vsebuje tudi posnemovalnik. Mogoča je tudi ločena inštalacija, se pa potem upravlja z ukazno vrstico, namesto preko grafičnega vmesnika. Ta lastnost je lahko tudi zaželjiva, saj orodja RN tudi nimajo grafičnih vmesnikov in je ravno tako potrebno uporabljati ukazno vrstico, kar poenoti interakcijo.

Za lažjo namestitev RN priporočajo namestitev Node.js, ker vsebuje paketni upravitelj Node.js Package Manager (NPM), s katerim se na preprost način upravlja namestitve in posodablja aplikacije na sistemih Windows. Orodja NPM se uporablja preko ukazne vrstice, kjer se sproži prenos programa, potem pa NPM samodejno poskrbi tudi za namestitev na računalnik. Ta program olajša pripravo projekta React Native, ker poskrbi za prenos, inštalacijo in konfiguracijo, da ima po zaključku razvijalec pripravljen delujoč projekt.

Najbolj vidna razlika med domorodnim in Reactovim razvojem je odsotnost grafičnega vmesnika pri nameščanju. Vse potrebno za domorodni razvoj je že zapakirano v enem paketu, namestitveni program pa uporabnika vodi čez celotno namestitev. V grafičnem vmesniku so funkcije bolj razumljivo razložene. Vsa orodja React Native pa se uporablja preko ukazne vrstice, kot je prikazano na sliki 4.1. Za lažjo namestitev je na spletu dokumentacija, ki uporabnike vodi korak za korakom od začetka do delujočega projekta.



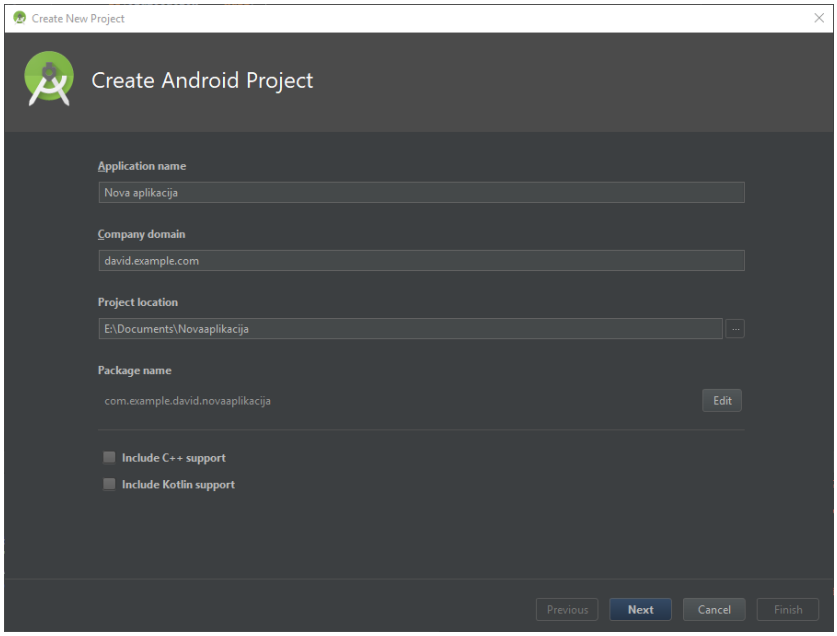
Slika 4.1: Uporaba orodij RN v ukazni vrstici.

4.2 Ustvarjanje projekta

Tako React Native kot Android Studio vsebujeta orodje za hitro pripravo novega projekta. V Android Studiu se lahko pri zagonu programa sproži postopek, ki uporabnika vodi skozi pripravo novega projekta. Aplikacija je nato pripravljena na razvoj in zagon.

Projekt React Native se pripravi v ukazni vrstici. Predpogoji so Python 2, Node, NPM in Java SDK. Z NPM se najprej naloži React Native CLI, orodja za upravljanje RN v ukazni vrstici. Rezultat je popolnoma pripravljen projekt React Native.

Razlika med Android Studiem in React Native je, da Studio uporabnika vodi čez postopek z grafičnim vmesnikom (primer okna na sliki 4.2), za React Native pa je potrebno navodila najti drugje, lahko v razdelku pomoči ukazne vrstice ali na spletu, kjer so navodila bolj obsežna, z opisom postopka in s slikami.



Slika 4.2: Okno priprave projekta.

4.3 Hierarhija datotek

Do določenega nivoja kompleksnosti ni potrebno poznati več kot samo nekaj osnovnih datotek in lokacij. Zavihek Projekt (*ang. Project*) v Android Studiu prikazuje vse datoteke, povezane z razvojem aplikacije – mapa *java* vsebuje programsko kodo, v *tests* so shranjeni avtomatizirani testi, v mapi *resources* pa ostale datoteke, ki jih potrebuje aplikacija, npr. slikovno gradivo. Naštete mape vsebujejo tudi podmape, katerih pomen ni razložen v programu, informacije o njih je potrebno zbrati drugje. Android SDK iz teh datotek ustvari aplikacijo, zato jih projekt React Native ravno tako vsebuje. Glavna datoteka za razvoj aplikacije Android je index.js. Za iOS obstaja ločena datoteka zaradi manjših razlik v programu in priporočenih oblikovalskih smernic, večina kode pa je lahko deljene med verzijama. To je glavna datoteka aplikacije, ki se izvede prva po zagonu in vsebuje inicializacijsko kodo za aplikacijo React Native. Prej omenjene datoteke ustvari React Native samodejno.

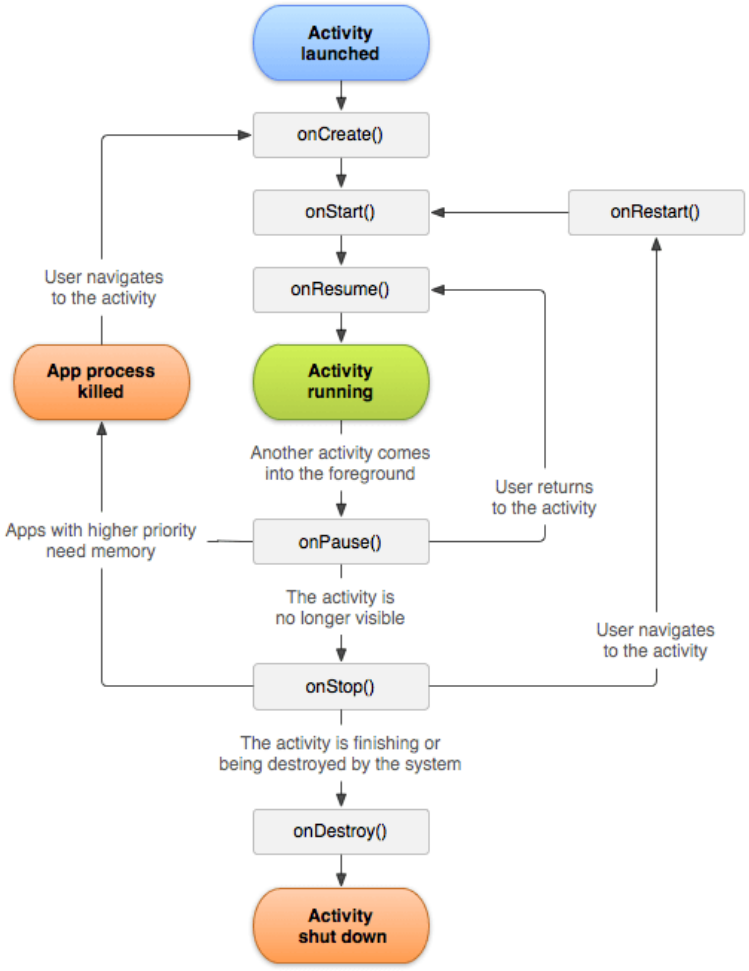
4.4 Poganjanje aplikacije in izdelava APK

Oba načina izdelave imata orodje za zagon razvojne verzije in izdelavo *Android Package* (APK). Za komuniciranje računalnika z napravami skrbi program *Android Development Tools* (ADT). Android Studio ima za ta program grafični vmesnik, v katerem so tudi izpisane naprave, tako da lahko razvijalec določi ciljno napravo, na kateri se zažene aplikacija. Če ni določena fizična naprava, se zažene posnemovalnik. React Native ravno tako omogoča hitri zagon, kot ponavadi se proces sproži v ukazni vrstici. V primeru, da fizična naprava ni povezana, se posnemovalnik ne zažene samodejno. Ponuja pa možnost, da so spremembe v kodi takoj vidne na napravi.

Brez certifikata se lahko ustvari samo nepodpisan APK, ki se ga uporablja za testiranje, vendar mora imeti naprava vklopljen razvojni način. Zaradi varnosti mora biti aplikacija podpisana s certifikatom, da uporabnik ne naloži nepreverjene aplikacije.

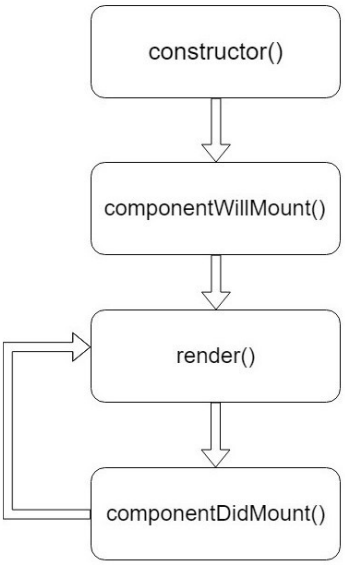
4.5 Struktura aplikacije

Glavni koncept pri domorodnem razvoju je aktivnost (*ang. Activity*), ki predstavlja posamezno okno aplikacije. Aplikacija jih lahko vsebuje poljubno število. Na sliki 4.3 je predstavljen življenjski cikel posamezne aktivnosti. Za spreminjanje delovanja je potrebno originalne metode povoziti in vstaviti lastno kodo. Aplikacijo se za delovanje pripravi v vstopni metodi *onCreate* v privzeti aktivnosti. V RN je njen najboljši približek metoda *componentDidMount*, vendar je zaradi usmerjenosti v oblikovanje življenjski cikel aplikacije v RN drugačen. Programska koda aplikacije se začne izvajati v *onCreate*, od koder se kličejo tudi ostale metode aplikacije.

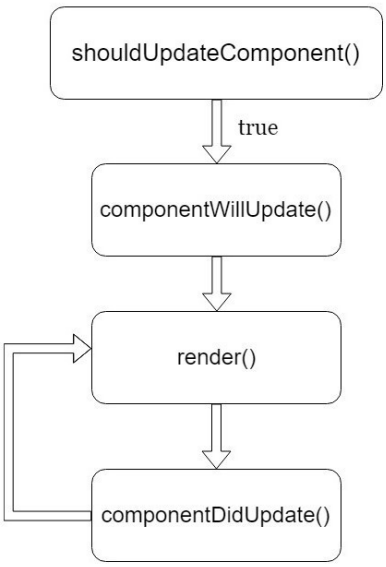


Slika 4.3: Življenjski cikel domorodne aplikacije. [1]

Orodja, ki vzpostavijo projekt React Native, pripravijo tudi vstopno datoteko aplikacije, vključijo vse nujne knjižnice (*React*in *Component*) in funkcije programa. Potek zagona aplikacije je orisan na sliki 4.4. Aplikacija prične izvajanje v metodi *constructor*, v kateri razvijalec inicializira potrebne podatke za zagon in delovanje. Pred prvim izrisom se izvede *componentWillMount*, kjer je omogočeno pridobivanje prvih zunanjih podatkov (recimo preko spleta). Metoda *render* poskrbi za izris na zaslon in se izvaja ves čas delovanja aplikacije, po prvi osvežitvi pa se enkrat izvede še *componentDidMount*, v katerem se lahko pripravi še zadnje podatke. Uporabna je tudi kot vstopna točka v aplikacijo. Po zaključku tega procesa se prične izvajati osveževalna zanka s slike 4.5. Ta proces je precej drugačen od domorodne aplikacije, ker se aplikacija osvežuje samo po potrebi in ne z določeno frekvenco. Metoda *render*, ki skrbi za ponoven izris na zaslon, se izvede šele ko metoda *shouldUpdateComponent* določi, da so v podatkih spremembe. Zatem pride na vrsto *componentWillUpdate*, v kateri lahko programer dodatno manipulira s podatki, preden se ponovno izrišejo.



Slika 4.4: Začetek izvajanja aplikacije React Native. [4]



Slika 4.5: Osveževalni cikel aplikacije React Native. [4]

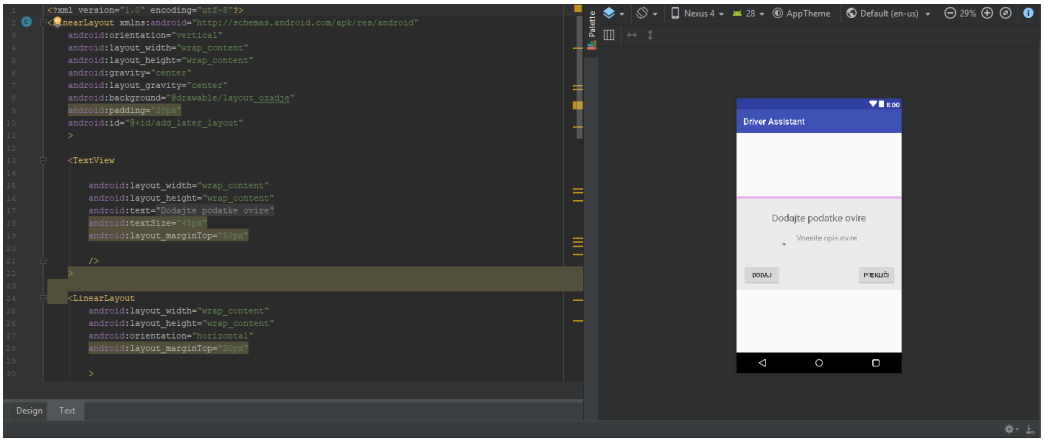
4.6 Oblikovanje

Android Studio vsebuje orodje za grafično oblikovanje aplikacije. Na sliki 4.6 sta vidna dva dela orodja. Grafični del na desni strani je uporaben za prototipe in grafično nezahtevne aplikacije, vsebuje vse tipe grafičnih elementov in razporeditve, da se lahko ustvari prototip postavitve. Sicer je oblikovanje definirano v datoteki XML, izpisani na levi strani, s specifičnimi značkami za Android.

V aplikaciji RN je oblikovanje definirano v funkciji *render*. To funkcijo ogrodje pokliče takrat, ko se pripravlja datoteko XML [2]. Funkcija mora vračati objekt tipa *element*, ki ga lahko razvijalec definira s kodo *Javascript as XML - JSX*. To je razširitev za sintakso JS, ki omogoča pisanje označevalne sintakse kar v kodi, brez potrebe po pakiranju v niz. Razširjena sintaksa se uporablja podobno kot HTML, kar je lahko bonus za spletne razvijalce, RN pa pri njegovi uporabi prikazuje boljša opozorila. Podobnost med JSX in HTML je vidna v sliki 4.7.

V stavkih JSX je mogoče uporabljati tudi program Javascript. Med zavite oklepaje se lahko vpiše ime spremenljivke, ime funkcije ali kar ukaz, iz katerega se pridobi vrednost. Stavki JSX se pri izgradnji pretvorijo v kodo Javascript, uradna dokumentacija pa pravi, da je kljub temu varno dovoliti uporabniške vnose, ker sistem vse vrednosti spremeni v niz znakov, zato vmešavanje v kodo programa ni mogoče. Pri kompilaciji se JSX pretvori v klice metode *React.createElement*, ta pa vrne objekt *React element*, ki vsebuje podane podatke in se uporablja za izgradnjo dokumentno objektnega modela (*ang. Document Object Model*) (DOM).

Elementi React se med izvajanjem ne morejo spreminjati. Če se mora vsebina spremeniti, mora razvijalec posodobiti želene elemente, potem pa poklicati metodo za osvežitev celotne podobe. V ozadju sistem poišče razlike in posodobi samo spremenjene elemente DOM. Filozofija tega sistema je, da vsaka posodobitev predstavlja celotno podobo v enem trenutku.



Slika 4.6: Datoteka activity.xml.

```
render() {

  return (
    <View style={styles.leftrightcontainer}>

      <View style={styles.leftContainer}>
        <View style={ styles.viewcontainer } >

        <TouchableOpacity style={{ height: 50, backgroundColor: "#616161" }}
            onPress={ () => { this.setState( { modalVisible: true } ) } }
          >
           <Text style={ { textAlign: 'center', fontSize: 36, color: 'white' } } >Report New</Text>
          </TouchableOpacity>

        <ListView
          dataSource={ this.ds.cloneWithRows(this.state.markers) }
          renderRow={(rowdata) =>
            <View style={ { flex: 1, width: 300, borderWidth: 2, flexDirection:'row' } } >
              {
                rowdata.rateable ? <TouchableOpacity style={{ backgroundColor: "red", flex: 1 }}
                  onPress={ () => { this.downvote( rowdata.key ) } }
                >
                <Text style={ { textAlign: 'center', fontSize: 36, color: 'white' } } >-</Text>
                </TouchableOpacity> : null
              }

              <View style={ { flex: 3, width: 300 } }>

                <View style={ { flex: 1, flexDirection:'row' } }>

                  <View style={ { flex: 1 } }></View>

                  <View style={ { flex: 2, alignItems: 'center'/*width: 100, height: 75*/ } }>
```

Slika 4.7: Metoda *render*.

4.7 Stylesheet

Ker je React samo knjižnica, ki se uporablja za spletne strani, se ravnokar opisan sistem uporablja samo za pripravo HTML, kjer je mogoče dodatno oblikovanje definirati s CSS, ki pa ga React Native ne podpira. RN zato ponuja *StyleSheet*. Podobnost s CSS je vidna na sliki 4.8. Ločena definicija oblikovanja izboljša berljivost kode in pohitri izvajanje, saj so stili shranjeni v objektih, ki se jih lahko ponovno uporabi. Stile se ustvari z metodo *StyleSheet.create*, ki za argument sprejme krovni objekt, ta pa vsebuje druge objekte, kateri predstavljajo selektorje. V selektorskih objektih se nahajajo polja, katerim razvijalec poda želene vrednosti.

Glavni navdih je pri obeh načinih razvoja isti, zato sta si filozofiji oblikovanja podobni. Različni sta si tudi v nekaterih bolj bistvenih delih (imena značk, različne datoteke, različni opisni jeziki), vendar ne dovolj, da bi razvijalec samo na podlagi oblikovanja izbral način razvoja, saj se v nekaterih bolj bistvenih delih razvoj domorodne aplikacije in aplikacije React Native bolj razlikujeta in lahko uporabnik napravi bolj smotrno odločitev.

```
const styles = StyleSheet.create({

  followButton: {
    height: 50,
    width: 50,
    alignSelf: 'flex-end',
    alignItems: 'flex-start',
  },

  centerPortion: {
    flexDirection:'row',
  },

  leftContainer: {
    flex: 1,
    flexWrap: 'wrap',
    alignItems: 'flex-start',

  },

  rightContainer: {
    flex: 1,
    flexWrap: 'wrap',
    //alignItems: 'flex-end',
    //borderWidth: 5,
    //justifyContent: 'flex-end',
  },
```

Slika 4.8: Objekt za oblikovanje.

4.8 Dovoljenja

Varovanje zasebnosti uporabnika je zelo pomembno. Za občutljive podatke je zato treba od uporabnika pridobiti dovoljenje, preden jih aplikacija lahko uporablja, za domorodni razvoj pa so na voljo API-ji. Pridobivanje pravic se je z verzijami spreminjalo. Do verzije Android 7.0 so morale biti uporabljene pravice zapisane v datoteki *AndroidManifest*, uporabniku se prikaže seznam pred inštalacijo in če se strinja, potem se aplikacija naloži in lahko prosto pridobiva vse dovoljene podatke. Če pa se uporabnik ne strinja, se aplikacija niti ne naloži. V zadnjih verzijah so poleg omenjenega načina (ki poskrbi tudi za kompatibilnost s starejšimi verzijami) dodali še zahtevo, da se pred prvo uporabo uporabnika ponovno povpraša po dovoljenju. Pri obeh so funkcije, ki poskrbijo za prikaz dialoga. Tudi v projektu React Native je potrebno urediti datoteko AndroidManifest, ki je enaka domorodni aplikaciji.

Razlika pri pridobivanju pravic v kodi je v tem, da je v funkciji RN potrebno definirati tudi obliko prikazanega dialoga, medtem ko funkcija v domorodni aplikaciji za argument sprejme samo zahtevane pravice.

4.9 Shranjevanje podatkov

Z RN je mogoče shranjevanje podatkov samo na en način, domorodna aplikacija pa jih ponuja več. V ozadju, izven kontrole programerja, se pri domorodni aplikaciji vsi podatki shranjujejo v lokalno podatkovno bazo (*RocksDB* ali *SQLite*, odvisno od podpore na napravi). V RN je na voljo modul *AsyncStorage*, ki deluje na principu ključ-vrednost. Dokumentacija predlaga, da se ta modul abstrahira za potrebe specifične aplikacije, ker deluje globalno (možna je okvara podatkov, saj so vsem aplikacijam dostopni vsi podatki). Za varnost podatkov je odgovoren programer, saj niso na noben način zaščiteni ali šifrirani. Ker je uporaba dolgoročnega pomnilnika dolgotrajen proces, tudi metoda ponuja asinhrono izvajanje – po klicu se glavni program normalno izvaja najprej, klicana funkcija pa deluje v drugi niti in ne upočasnjuje aplikacije, po zaključku pa izvede dodeljeno funkcijo. Domorodni razvoj po drugi

strani ponuja več metod za različne potrebe aplikacije. Ekvivalent RN ponuja razred *SharedPreferences*. Na voljo sta dve metodi, ena za argument potrebuje samo podatke, drugi pa se lahko poda še ime datoteke, kamor se podatki shranjujejo. V nasprotju z RN so uporabljene datoteke dostopne samo aktualni aplikaciji, zato ni nevarnosti za poškodovanje podatkov. Datoteke morajo biti ustvarjene pred uporabo in zapisane v AndroidManifestu. Za zapisovanje in branje je potrebno uporabiti metode, primerne za tip podatka, npr. *putInt* in *getInt* za celoštevilske vrednosti. Te metode ne podpirajo večnitnega izvajanja samodejno, ampak mora za to poskrbeti programer.

4.10 Senzorji

V nasprotju z RN je pri domorodnem razvoju uporaba senzorjev dobro podprta. S senzorji se lahko upravlja samo preko zunanjih knjižnic, zato je koda za vsak senzor različna in jih bomo opisali ločeno. Zaradi raznolikosti naprav je potrebno najprej določiti, če je senzor na voljo, kar je odvisno od naprave (če vsebuje strojno opremo) in verzije operacijskega sistema [3]. Ti podatki se pri domorodni aplikaciji nahajajo v instanci objekta tipa *SensorManager*, ki se ga pridobi od operacijskega sistema. Ta objekt vsebuje vse metode za delo in vse podatke o razpoložljivih senzorjih, seznam pa je mogoče tudi omejiti na določene tipe. Vsak tip ima različne podatke in razpone vrednosti, zato je delo z vsakim senzorjem drugačno, vendar razred ponuja enovit način dostopa do teh informacij. Podatke se lahko pridobi ob eksplicitni zahtevi, ob spremembah pa lahko prožijo tudi vrnitvene metode.

4.11 Orientacija

V času izdelovanja aplikacije React Native ni ponujal kontrole nad obračanjem zaslona, je pa na spletu mogoče dobiti modul *react-native-orientation*. Najlažje se ga namesti z NPM-jem, tako da je po sprožitvi ukaza modul že pripravljen za uporabo brez dodatnega konfiguriranja. Modul vsebuje pet metod, štiri od teh zaklenejo orientacijo, ena pa omogoči prosto obračanje. Poleg teh metod sta še dva dogodka, ki se sprožita ob spremembi orientacije. Domorodne funkcije so bolj robustne, zaklep orientacije je mogoč na dva različna načina – v datoteki XML za oblikovanje aktivnosti in v programski kodi. Spreminjanje datoteke XML je najhitrejši način, vendar nefleksibilen, saj naknadno spreminjanje v kodi ni omogočeno. V programski kodi se doseže podobne rezultate kot s prej omenjenim dodatkom v RN, vendar je programska koda bolj zapletena, saj ima več funkcionalnosti in je bolj splošnonamenska.

4.12 Lokacija

Lokacija je ena izmed informacij, za katere je potrebno imeti dovoljenje uporabnika. Obstajata dve različni lokacijski storitvi, *Coarse location* (groba) in *Fine location* (natančna). Razlikujeta se v natančnosti, groba lokacija uporablja samo mobilni signal, natančna lokacija pa tudi senzor GPS. Obe storitvi imata ločeni dovoljenji. Tako domorodni kot razvoj RN imata enak potek pridobivanja lokacijskih podatkov, le malo se razlikujeta v manjših podrobnostih, npr. v imenih razredov. Pri obeh je potrebno pred zahtevo lokacije preveriti, če je uporabnik podal dovoljenje za uporabo. Aplikacija bo ravno tako delovala brez tega preverjanja, vendar če nima dovoljenja, mu v naslednjem koraku, ko se zahteva podatke, funkcija ne bo vrnila podatkov. Preverjanje je potrebno tudi zato, da se lahko uporabniku pred prvo uporabo ponudi dialog za dovoljenje. Razlika pa se pokaže pri funkciji za pridobivanje lokacije. Pri domorodni aplikaciji je pridobivanje preprostejše, saj se uporabi dve funkciji, vsako za eno koordinato (obstaja tudi funkcija ki vrne zadnjo znano lokacijo, za boljšo odzivnost). Pri React Native je po-

stopek malo drugačen zaradi stanja. Funkcija za pridobivanje lokacije vrača rezultate v vrnitveni funkciji, ki se izvede ob vsaki zaznani spremembi koordinat, medtem ko mora pri domorodnem razvoju programer sam poskrbeti za osveževanje vmesnika.

4.13 Kompas

Še ena izmed trenutno nepodprtih funkcionalnosti v RN je pridobivanje usmerjenosti naprave oziroma kompas. Ta funkcionalnost je dostopna v paketu *react-native-simple-compass*. Paket doda dve novi funkciji. Z eno se sproži sledenje obračanju, ki s prvim argumentom določi, ob kolikšni razliki v stopinjah se sproži vrnitvena funkcija, ki je podana v drugem argumentu. Pridobivanje orientacije v domorodni aplikaciji je spet bolj zapleteno, ker funkcija nima ovojnice, ki na poenostavljen način vrača želene vrednosti. API ponuja dostop do veliko različnih senzorjev (merilnik pospeškov, magnetomer, žiroskop ...), s katerimi je mogoče na različne načine in z različno natančnostjo pridobiti usmerjenost. Senzor tipa *TYPE_ROTATION_VECTOR* ob spremembi meritev sproži dogodek, kateremu dostavi štiri decimalna števila, ki skupaj predstavljajo usmerjenost naprave v obliki kvaternionov. Tretja vrednost po vrsti v tem objektu predstavlja rotacijo okrog osi Z, oziroma kot med usmerjenostjo naprave in severom.

4.14 Vibracija

Vibracije so v RN podprte z asinhrono funkcijo. Pred uporabo je treba v datoteko AndroidManifest dodati dovoljenje. Glede na argumente ima različne rezultate - celoštevilski argument bo določil čas enakomernega vibriranja v milisekundah, seznam celih števil definira vzorec (liha števila so premori, soda števila čas vibracije), zadnji argument pa omogoči ponavljanje vzorca do preklica. Domorodna aplikacija vsebuje nekaj korakov več, ker je potrebno od sistema najprej pridobiti storitev za vibriranje. Proženje vibracij je odvisno

od nivoja API (verzija sistema Android). Do verzije 26 se uporablja metoda *vibrate*, uporaba je zelo podobna RN. Celo število določi dolžino vibriranja, seznam števil pa vzorec. Majhna razlika je v tem, da se ponavljanje vzorca določi z indeksom začetka ponovitev v podanem seznamu. V novih verzijah je potrebno namesto surovih vrednosti uporabiti razred *VibrationEffect* (ki se ga instancira z želenimi vrednostmi).

4.15 Komunikacija

Oba načina razvoja omogočata komunikacijo s strežnikom REST brez dodatnih knjižnic. Strežnik REST sprejema zahteve POST, ki so standardizirane, zato so podani podatki zmeraj enaki, imata pa drugačne postopke in metode. V sklopu *headers* se lahko zahtevku POST doda tudi druge informacije v obliki ključ-vrednost, ki jih prejme strežnik, na primer avtorizacijski podatki (uporabniško ime in geslo za dostop do strežnika) pod ključem *Authorization* in oblika vrnjenih podatkov (npr. JSON) pod ključem *Accept*. React Native za komunikacijo s strežnikom ponuja funkcijo *fetch*, ki za argumente sprejme naslov URL in objekt, v katerem so nastavljene prej omenjene vrednosti. Izvaja se v ločeni niti, da je aplikacija odzivna (komunikacija preko spleta je relativno počasna). Ko se zaključi, se izvede vrnitvena metoda, podana v argumentu. Rezultat je generičen in neberljiv objekt z odzivom strežnika, ki se ga lahko pretvori v objekt, strukturiran kot JSON, da se na lažji način dostopa do podatkov. V domorodni aplikaciji je postopek zelo podoben, samo da se izkorišča funkcionalnost razredov – v RN se pokliče funkcijo *fetch*, kateri se poda potrebne argumente, v Javi pa je potrebno ustvariti objekt (tipa *HttpGet*), ki v konstruktorju zahteva naslov URL strežnika (kot v RN), od tukaj naprej pa se manipulira s tem novim objektom. Dodatne podatke se zahtevku dodeli s funkcijo, argumenta sta ključ in vrednost. Ta funkcija ustvari objekt, ki se poda kot drugi argument funkciji *fetch*. Za komunikacijo s strežniki se uporablja objekt *DefaultHttpClient*, ki tudi skrbi za sprejem odziva. Celoten rezultat s strežnika se shrani v objekt, za lažjo

uporabo pridobljenih podatkov pa se ga lahko pretvori v drug razred, ki izpusti velikokrat neuporabne podatke. Ta objekt potem vsebuje niz z vsebino prejetega rezultata JSON.

4.16 Testiranje

Aplikacijo se testira na enak način, če je domorodna ali narejena z RN. Razvijalci imajo na izbiro fizično napravo (npr. mobilni telefon ali tablico) in posnemovalnik (na sliki 4.9 je prikazan posnemovalnik v Android Studiu). Posnemovalnik je program, ki omogoča izvajanje programov enega operacijskega sistema na drugem. Z Googlovim posnemovalnikom je mogoče v Studiu (za domorodne aplikacije) in z orodji RN pognati aplikacijo na osebnem računalniku brez izdelave inštalacijskega paketa. V ADT je viden kot fizična naprava. To tudi pomeni, da lahko tudi fizično napravo uporabljamo na enak način. V Android Studiu je tudi mogoče z enim klikom pognati aplikacijo na napravi.

Slika 4.9: Posnemovalnik.

4.17 Primerjava aplikacij

Aplikacija ni dovolj zahtevna, da bi uporabniki lahko opazili zmogljivostne razlike, s stališča izkušnje sta obe praktično enaki z zelo majhnimi razlikami. En pomemben kriterij za uporabnike je velikost aplikacije (recimo zaradi omejenih količin podatkov ali nepotrpežljivosti uporabnika). Večje razlike bo opazil razvijalec, zato se bomo bolj osredotočili na to plat.

Pri obeh načinih razvoja je mogoče doseči enako delovanje, pot do tega cilja pa se pri določenih komponentah razlikuje. Razlike v delovanju aplikacije so za uporabnika sicer transparentne. Ena tehnika za dosego paritete funkcionalnosti v aplikaciji React Native je uporaba domorodnih klicev. Precej funkcionalnosti v RN trenutno manjka, a jih je mogoče pridobiti s knjižnicami, ki jih ustvarjajo zanesenjaki in so dostopne na spletu. Tudi te zunanje knjižnice so domorodna koda, katere se kliče iz programa Javascript.

Za knjižnice ne obstaja osrednji portal, kjer bi bile zbrane, ampak jih je potrebno najti vsako posebej. Vse so odprtokodne in ponavadi shranjene na različnih spletnih repozitorijih (npr. GitHub), kjer so napisana tudi navodila za uporabo. Večino se naloži kar preko ukazne vrstice z NPM. Zanimiv stranski učinek zunanjih knjižnic je, da v veliko primerih olajšajo uporabo senzorjev, saj bolj zapleteno domorodno kodo zapakirajo v preprostejše funkcije.

Programerji lahko opazijo razliko pri količini napisane kode, kar je lahko en kriterij za določanje zahtevnosti programa. Sam po sebi ni zanesljiv, vendar lahko skupaj z drugimi oriše približno zahtevnost projekta. Domorodni projekt je spisan v približno 1650 vrsticah, React Native pa v približno 600 vrsticah. Nobena od teh dveh ni optimizirana. Program React Native je krajši, ker so funkcije preprostejše za uporabo. Veliko več se uporablja statične razrede, ki direktno dostopajo do funkcionalnosti naprave, medtem ko se v domorodnem programu praviloma pridobiva reference do funkcionalnosti od operacijskega sistema. Zaradi bolj kompleksnih verig referenc so stavki daljši in vsebujejo več klicev metod, bolj kompleksno pa je tudi upravljanje vmesniških komponent (programer mora skrbeti za zaznavanje

interakcije in dinamično spreminjanje elementov), kjer je RN preprostejši, saj je prirejen programiranju vmesnikov. Sicer v zmožnostih obeh ni razlike. Domorodna aplikacija omogoča najbolj nizkonivojski dostop, zato ima teoretično največjo zmogljivost. Vseeno pa orodja React Native za oblikovanje niso nič manj zmogljiva, dodaten plus za nekatere razvijalce pa je še podobnost s HTML in CSS, ki sta že dolgo standard na spletu.

Problematična točka za programerje je lahko tudi čas izdelave aplikacije, ker je lahko čakalni čas do izdelanega paketa zelo dolg in bi upočasnil razvoj. V primeru te aplikacije se to ni zgodilo. Je pa tudi v tej aplikaciji opazna razlika. Najdaljši časi so pri vzpostavitvi projekta in prvem zagonu, saj se takrat ustvarijo še preostale potrebne datoteke za aplikacijo in so lahko dolgi po več minut. Daljši od povprečja so tudi ob vsakem zagonu, ko programa ni v pomnilniku. Pri domorodnem razvoju so časi konsistentno krajši. Rezultati so zapisani v tabeli. Odstopajo po več sekund, zelo je odvisno od trenutne obremenjenosti računalnika (programi v ozadju, protivirusni programi, orodja v sistemu Windows, spletni brskalniki). Povprečen čas pri RN je 9x daljši kot pri domorodnem razvoju. Daljši časi so posledica dejstva, da se mora projekt RN najprej pretvoriti v domorodni projekt, šele potem se lahko zgradi aplikacijo. Zanimiv podatek pri RN je, da se ob ponovnem zagonu čas zagona na emulatorju ne razlikuje bistveno od povprečja. Razlika je opazna tudi pri izdelavi paketa APK, vendar je manjša. Poudariti je treba, da so časi zelo odvisni od strojne opreme, obremenjenosti računalnika in zahtevnosti programa. Pridobljene vrednosti nakazujejo samo trend, da je React Native v povprečju počasnejši od orodij v Android Studiu. Za bolj natančne zaključke je potrebna analiza več aplikacij.

Rezultati so bili pridobljeni na namiznem računalniku z naslednjimi specifikacijami:

- Procesor: Intel i7-870 2.93GHz
- Pomnilnik: 4 GB DDR3 1333 MHz
- Trdi disk: Samsung 850 Evo 250 GB (SSD, SATA II priključek)
- Matična plošča: Asus P7H55-M SI
- Grafična kartica: AMD Radeon R9 380 4 GB (Asus)

Ogrodje	**Čas 1**	**Čas 2**	**Čas 3**	**Povprečje**
Android Studio	**3,696 s**	**4,210 s**	**1,766 s**	**3,224 s**
React Native	**29,324 s**	**28,703 s**	**31,014 s**	**29,680 s**

Tabela 4.1: Časi izdelave APK

	Android Studio	**React Native**	**Absolutna razlika**	**Povečanje v odstotkih**
Povprečen čas izdelave APK	**3,224 s**	**29,680 s**	**26,456 s**	**920,596 %**
Velikost APK	**4,480 MB**	**8,124 MB**	**3,644 MB**	**181,250 %**

Tabela 4.2: Primerjava časov in velikosti

Prva stvar, ki jo opazi uporabnik je velikost aplikacije, kjer je RN ponovno bolj potraten, vendar v manjši meri kot pri času izdelave. Večji je za približno 4 MB. Aplikacija ni dovolj velika, da bi uporabnik imel probleme zaradi porabe prostora, morajo pa razvijalci upoštevati, da je na Google Play postavljena meja pri 100 MB, ob presežku katere se prikaže opozorilo o velikosti aplikacije. Aplikacije, ki presegajo to mejo (tudi če samo za nekaj kilobajtov), imajo precej nižje število prenosov. Razlika pri RN je v redkih primerih dovolj velika, da potisne aplikacijo čez mejo.

Za primerjavo, osnovna aplikacija samo z dvema vrsticama teksta je velika 7,794 MB, če je ustvarjena z RN verzije 0.52, v kateri je narejena tudi aplikacija Driver Assistant. V kasnejših verzijah se je velikost povečala, v verziji 0.58 na 16,612 MB. Podobno velika je tudi v ogrodju Expo, ki se uporablja v zadnji verziji RN. Expo so dodatna orodja za razvoj v React Native, ki pridodajo tudi k velikosti aplikacije. Velikost je mogoče zmanjšati, na primer če se loči verzije za procesorje x86 in ARM (večino naprav Android poganja procesor ARM). Posledično se ustvarita dva paketa APK, paket ARM pa je manjši, saj ne vsebuje kode za x86 naprave. Druga možnost je uporaba orodja kot je *ProGuard*, ki iz programa oklesti neuporabljeno kodo in knjižnice.

Razlike je treba pravilno interpretirati. V primeru te aplikacije so se razlike merile v večkratnikih, vendar velik del dodatnega časa in prostora uporabijo orodja RN (na primer za interpretiranje Javascripta v Javo). Ta orodja so vedno enake velikosti, ampak pri tako majhni aplikaciji predstavljajo velik delež.

4.18 Odzivnost

Pri uporabi ni zaznavnih razlik. Čas zagona je zelo kratek in je enak pri obeh verzijah. Prva dlje trajajoča zadeva je nalaganje podatkov mape, ki pa je odvisna od hitrosti interneta. Edina razlika, ki jo uporabnik opazi, je nalaganje slik. Element, ki prikazuje slike v RN, samodejno poskrbi za nalaganje v drugi programski niti, zato se program ne zaustavi med čakanjem. Se pa zato slike prikažejo na zaslonu čez čas in ne hkrati. Programer lahko prepreči dinamično nalaganje in prikaže vse slike naenkrat, če določi, da program počaka vse slike, preden prikaže nov dialog. V domorodnem programu je to privzeto delovanje, ker se nalaganje slik ne izvaja vzporedno. Programer mora ročno poskrbeti, da se sproži vzporedna nit, v kateri se nalagajo slike. Gumbi so enako odzivni, komunikacija s spletnim strežnikom traja enako dolgo na istih povezavah ... S stališča odzivnosti in uporabniške izkušnje razlike niso dovolj velike, da bi se na njihovi podlagi odločili za eno izmed orodij, saj je mogoče z boljšo programsko kodo izenačiti delovanje obeh.

4.19 Povzetek

Rezultati primerjave so povzeti v tabeli 4.3. Tabela je razdeljena po komponentah aplikacije, pri vsaki pa je razloženo, kateri način razvoja ima uporabo bolje implementirano. Razvijalec lahko izbere React Native ali domorodni razvoj glede na število kategorij, ki imajo boljšo rešitev pri najbolj pomembnih zahtevah aplikacije in razvijalcev, npr. če imajo oblikovalci veliko izkušenj s spletnimi tehnologijami ali pa če ima aplikacija zapleten in dinamičen vmesnik, potem je boljša izbira RN.

Kategorija	Boljša rešitev	Razlog
Namestitev orodij	Domorodna aplikacija	Za pripravo domorodnega razvojnega okolja je dovolj prenos enega paketa, za React Native se jih potrebuje več.
Ustvarjanje projekta	Domorodna aplikacija	Grafična orodja vodijo skozi celoten postopek, React Native potrebuje zunanjega vodiča in ročno pisanje ukazov.
Poganjanje aplikacije in izdelava APK	Domorodna aplikacija ali aplikacija React Native	Pri obeh je skorajda enako preprosto pognati aplikacijo za testiranje, Android je malo lažji zaradi gumba, React Native pa ima prednost v tem, da se lahko aplikacijo posodobi med izvajanjem.
Oblikovanje	React Native	Zaradi podobnosti s spletnimi tehnologijami, ki so močne in splošno znane
Pridobivanje dovoljenj	Domorodna aplikacija ali aplikacija React Native	Postopek je podoben, se pa potrebuje zunanjo knjižnico za React Native
Shranjevanje	React Native	React Native potrebuje dodatno knjižnico, ki pa so lažje za uporabo kot pisanje domorodne kode
Uporaba senzorjev	Domorodna aplikacija ali aplikacija React Native	React Native potrebuje dodatne knjižnice za vsak senzor posebej, je pa potem dostop do senzorjev preprostejši
Spletna komunikacija	Domorodna aplikacija ali aplikacija React Native	Praktično enaka uporaba
Končna aplikacija	Domorodna aplikacija ali aplikacija React Native	Zelo odvisno od potreb in razvijalcev, bolj natančno opisano drugje

Tabela 4.3: Pregled primerjave

Poglavje 5

Zaključek

Za primerjavo smo razvili dve inačici aplikacije. Primerjava je razdeljena na razvojni in uporabniški del, ki sta še naprej razdeljena na kategorije. Pri razvojnem delu so primerjana IDE in orodja, ki jih vsebujeta Android Studio in React Native, ter posamezne komponente aplikacije (recimo spletna komunikacija in shranjevanje podatkov). Pri uporabniškem delu sta primerjani končni aplikaciji in njuni deli - velikost, izgled, funkcionalnost, odzivnost ...

Če povzamemo, bi domorodni razvoj priporočali razvijalcem, ki bolje poznajo Javo kot Javascript, ali če je aplikacija strojno zahtevna (ker so lahko bolj optimizirane) ali pa zaradi boljših orodij, ker ni časa za učenje RN. React Native je bolj primeren za spletne razvijalce zaradi Javascripta in sistema oblikovanja, več podprtih operacijskih sistemov in za aplikacije, kjer se informacije pogosto spreminjajo. Za zelo neizkušene programerje je bolj primeren razvoj domorodne aplikacije, ker so orodja veliko bolj prijazna do začetnikov, predvsem zaradi vodiča in grafičnega vmesnika. Za implementacijo nekaterih komponent, npr. pridobivanje dovoljenj, shranjevanje, orientacijo ali uporabo senzorjev, je pri aplikaciji RN potrebna zunanja knjižnica. Vložen čas učenja knjižnice se nato ponavadi poravna z lažjo uporabo.

Z RN je mogoče ustvariti enako aplikacijo po izgledu in funkcionalnosti, razlika je v velikosti inštalacijskega paketa, kar vpliva na zelo majhne aplikacije, ali pa na tiste, ki so blizu meje, kjer Google Play vpraša za vklop

brezžičnega interneta. Odločitev je zato bolj odvisna od specifik razvoja in aplikacije. Zelo pomembno je znanje programerjev, ker na razvojni čas zelo vpliva znanje uporabljenih orodij. Kljub podobnemu imenu sta si Java in Javascript precej različna, saj sta namenjena različnemu delu. Java je bolj splošnonamenska, Javascript pa je bolj usmerjen v spletne tehnologije. Zaradi teh razlik v filozofiji lahko drugače vajenim programerjem ne ustreza. Drugačna so tudi programerska orodja in okolja, kar ima podobne posledice.

Naslednje vprašanje se tiče potreb projekta. React Native ima dve specifiki. Prva je podpora za več operacijskih sistemov. Z majhnimi spremembami v kodi se lahko ustvari aplikacijo za Android in iOS hkrati. V primeru, da projekt nima dovolj časa ali izurjenih programerjev za razvoj dveh ločenih domorodnih aplikacij, se lahko privarčuje veliko časa. Druga posebnost je usmerjenost v uporabniške vmesnike, kar skupaj s programskim konstruktom *state* precej olajša delo z vmesnikom. Celotno ogrodje je zasnovano okrog stanja, v katerem so shranjeni vsi podatki aplikacije, uporabniški vmesnik pa se posodobi samodejno. Nekatere aplikacije, ki uporabljajo veliko podatkov, ki se pogosto spreminjajo, lahko to sposobnost dobro izkoristijo.

Zmožnosti RN so tako skorajda enakovredne domorodnemu razvoju v Javi, manjkajo pa določene funkcionalnosti, ki se jih pridobi z knjižnicami, vendar jih je treba najprej najti. Večinoma so preproste za uporabo, ker imajo zelo ozko namembnost. Sicer pa je mogoče tudi klicanje domorodnih metod iz programov RN.

Razvoj domorodne aplikacije je v nasprotju bolj tehnično zahteven. Java je bolj rigidna, zaradi nižjega dostopa je potrebne več kode za enako delovanje. To se najbolj opazi pri upravljanju z vmesniki, kjer je kode občutno več in je manj intuitivna. Druga plat zahtevnosti pa so manjše aplikacije in hitrejše delovanje, saj je preprostost React Native rezultat več slojev abstrakcije. Druga plat domorodnega razvoja pa so bolj dodelana orodja, ki jih je lažje uporabljati kot ukazno vrstico, preko katere se upravlja z React Native. Android Studio uporabnika vodi čez proces vzpostavitve projekta, nalaganja posnemovalnika, razvojnih orodij, ustvarjanje paketov itd. React Native niti

ne vsebuje vseh ravnokar naštetih orodij, ostale pa se sproži s tekstovnimi ukazi, ki jih mora uporabnik spoznati nekje drugje. Programerska okolja pa si uporabnik izbere sam, zato je tukaj primerjava težka.

Literatura

[1] Android Activity Lifecycle. Dosegljivo: https://www.javatpoint.com/android-life-cycle-of-activity. [Dostopano: 16.1.2019].

[2] Introducing JSX. Dosegljivo: https://reactjs.org/docs/introducing-jsx.html. [Dostopano: 16.1.2019].

[3] Sensors Overview. Dosegljivo: https://developer.android.com/guide/topics/sensors/sensors_overview. [Dostopano: 16.1.2019].

[4] Understanding React Native Component Lifecycle Api. Dosegljivo: https://blog.usejournal.com/understanding-react-native-component-lifecycle-api-d78e06870c6d. [Dostopano: 16.1.2019].